Second-order Variables Explained Away

Thomas Ede Zimmermann

0 Introduction

It is well-known that first-order predicate logic can be given a variable-free or combinatory formulation. However, until fairly recently, no attempts seem to have been made to find an adequate set of combinatorial operators for second-order logic. This may partly be so, because the task was felt to be both unrewarding and straightforward; for in spite of its semantic complexity, the expressive means and mechanisms of second-order logic are so similar to those of first-order logic, that a combinatorial reformulation can only be a slight (if messy) adaptation of, for example, Quine's (1960) *Predicate Functor Logic* (*PFL*).[1] This view has been challenged by Kosta Došen (1988) who, in his attempt to formulate a variable-free version of second-order logic, came to the conclusion that this task calls for new predicate functors, most of which 'play a role analogous to the role of the old combinatory predicate functors', but some of which 'play a completely new combinatory role' (245). In this paper I will give a variable-free formulation of second-order logic whose logical operators are all straightforward adaptations of Quine's predicate functors. So in the end, we will have turned the old bias against second-order PFL into a well-founded judgement.

The paper is organized as follows: In section 1, I will sketch what I take to be the most natural *semantic* motivation for reformulating predicate logic in just the way Quine (1960) did. These considerations will turn out to be directly transferrable to second-order logic, whose combinatorial formulation will be discussed in section 2. The third section briefly looks at two alternative formulations of second-order PFL, one of which is Došen's (1988).

1 From a Semantic Point of View

1.1 Compositionality in Predicate Logic

In (classical) propositional logic, things are simple. Formulae denote truth-values; atomic formulae do it in an arbitrary way, whereas the denotations of complex formulae

[1] This is not the only variable-free formulation of first-order logic: two variations can be found in Bernays (1959) and Quine (1974: 380f.); see Bacon (1985) for some historical background, and a lot more on the subject. To keep things compatible with Došen (1988), I am concentrating on the Quine (1960) version (though the notation is my own); but whatever I am saying about the relation of first and second order equally applies, *mutatis mutandis*, to the aforementioned variants (and others).

systematically depend on the logical material they contain: $[\![\neg\varphi]\!]^M = \overline{[\![\varphi]\!]^M}$, $[\![\varphi\&\psi]\!]^M = [\![\varphi]\!]^M \times [\![\psi]\!]^M$. Propositional logic is thus compositional in that it is interpreted by homomorphisms from its syntax (*Fml*; *Neg*, *Conj*) to its semantics ($\{0,1\}$; $\overline{\ }$, $\times$).

Things, however, are less simple with (first-order) predicate logic. For if we insist that formulae denote truth-values, then the semantics of predicate logic appears to be non-compositional: '$(\exists x)\varphi$' and '$(\exists x)\neg\varphi$' may both be true, even though either φ or '$\neg\varphi$' is false and thus coincides in its truth-value with, for example, '$(\varphi\ \&\ \neg\varphi)$'; but, clearly, '$(\exists x)(\varphi\ \&\ \neg\varphi)$' is always false, so that no operation could determine the truth-value of '$(\exists x)\varphi$' (or '$(\exists x)\neg\varphi$') from that of φ (or '$\neg\varphi$').

If truth-values do not suffice for giving a compositional interpretation to predicate logic, it may be the case that something else does. Indeed, it seems quite plausible to assume that first-order formulae express relations among individuals, in the sense in which

(C) $\qquad (\exists y)[R(x,y)\ \&\ S(y,z)]$

expresses the composition of the binary relations (expressed by) R and S. More generally, given a model M of (a particular language of) first-order logic, and a formula φ (of that language) containing n free variables, we may say that f expresses (in M) the following n-ary relation $[\![\varphi]\!]^M$ on the universe U_M of M:

$$[\![\varphi]\!]^M = \{(u_1,\ldots,u_n) \in {U_M}^n \mid M \models {}^{x_1,\ldots,x_n}_{u1,\ldots,un}\,\varphi\},$$

where the variables are listed in the order of their first appearance in φ.[2] So, can we formulate a compositional interpretation of predicate logic in terms of the relations expressed by its formulae?

Let us first consider negation. Do we have $[\![\neg\varphi]\!]^M = \mathfrak{N}([\![\varphi]\!]^M)$, for some global logical operation $\mathfrak{N}$? The answer is obviously positive if we let $\mathfrak{N}_n(R)$ be the complement of the n-ary relation R (relative to ${U_M}^n$):

$$[\![\neg\varphi]\!]^M = \{\vec{u} \mid M \models^{\vec{x}}_{\vec{u}} \neg\varphi\} = \{\vec{u} \mid M \not\models^{\vec{x}}_{\vec{u}} \varphi\} = \mathfrak{N}_n(\{\vec{u} \mid M \models^{\vec{x}}_{\vec{u}} \varphi\}) = \mathfrak{N}_n([\![\varphi]\!]^M).$$

Next consider existential quantification. The relation expressed by a formula like '$(\exists y)R(x,y)$' depends on $[\![R(x,y)]\!]^M$ in a very straightforward way:

$$[\![(\exists y)R(x,y)]\!]^M = \mathfrak{E}_2([\![R(x,y)]\!]^M),$$

where $\mathfrak{E}_2$(R) is the projection $\bigcup_v \{(u,v) \in R\}$ of the binary relation R. However, a slightly different operation is involved in determining, for example, $[\![(\exists x)R(x,y)]\!]^M$; and yet more variants of $\mathfrak{E}_2$ are needed for formulae containing more than one free variable. It thus seems that existential quantification corresponds to a whole family of operations, and which one applies in a given case depends on the exact nature of the sub-formulae. But clearly, there is a common core to all of these operations, in that they are all suitable n-place generalizations $\mathfrak{E}_n$ of $\mathfrak{E}_2$ acting on the result of inverted n-place relations: whenever φ is a formula with free variables $x_1,\ldots,x_n$ (in that order), then

[2] The notation should be self-explanatory. – One could also have $[\![\varphi]\!]^M$ depend on the *standard order* $x_1, x_2, x_3,\ldots$ of all individual variables, rather than their appearance in φ. However, some of the operations to be defined below would then turn out to be more awkward. Still, the standard order will be of use for the translation of *PFL* into predicate logic: see section 1.2.

$$[\![(\exists x_i)\varphi]\!]^M = \mathfrak{E}_n(\mathfrak{I}_{i,n}([\![\varphi]\!]^M)),$$

where the inversion $\mathfrak{I}_{i,n}$ is defined by:

$$\begin{aligned}\mathfrak{I}_{i,n}(R) &= \{(u_1,\ldots,u_{i-1},u_{i+1},\ldots,u_n,u_i)|(u_1,\ldots,u_n)\in R\}, \text{ and:}\\ \mathfrak{E}_n(R) &= \bigcup_{u_n}\{(u_1,\ldots,u_{n-1})|(u_1,\ldots,u_n)\in R\}.\end{aligned}$$

The $\mathfrak{I}_{i,n}$ can be seen as upgraded permutations on n-tuples, so that we may think of them as being composed out of the basic inversions $\mathfrak{J}$(ump) and $\mathfrak{S}$(wap), corresponding to certain basic permutations:

$$\begin{aligned}\mathfrak{J}_n(R) &= \{(u_2,\ldots,u_n,u_1)|(u_1,\ldots,u_n)\in R\}\\ \mathfrak{S}_n(R) &= \{(u_1,\ldots,u_n,u_{n-1})|(u_1,\ldots,u_n)\in R\}\end{aligned}$$

So an existential quantifier prefix to a formula φ with n free variables, can be interpreted as an operation of the form $\mathfrak{E}_n \circ \mathfrak{I}_{i,n} = \mathfrak{E}_n \circ (\mathfrak{S}_n \circ \mathfrak{J}_n)^{n-i} \circ \mathfrak{J}_n{}^i$ on the relation expressed by φ.[3] This is as close as we can get to a compositional interpretation of quantification, using relations as meanings.

Finally consider conjunction. Again, there appears to be a natural candidate for its compositional interpretation, viz. the *product* (or lifted concatenation) operation[4] $\mathfrak{A}_{n,m}$ defined on (n-ary and m-ary) relations R and S:

$$\mathfrak{A}_{n,m}(R,S) = \{(u_1,\ldots,u_n,v_1,\ldots,v_m)|(u_1,\ldots,u_n)\in R \text{ and } (v_1,\ldots,v_m)\in S\}$$

Clearly, $\mathfrak{A}_{n,m}$ can be used to combine the relations expressed by φ and ψ in case they have no free variables in common. But in general, $\mathfrak{A}_{n,m}$ would have to be slightly adapted[5]: if

$$Fr(\varphi) = \{x_1,\ldots,x_n\},\ Fr(\psi) = \{y_{n+1},\ldots,y_{n+m}\},\ Fr(\varphi)\cap Fr(\psi) = \{x_{i1},\ldots,x_{ik}\} = \{y_{j1},\ldots,y_{jk}\},$$

and $Fr(\psi)\backslash Fr(\varphi) = \{y_{j_{k+1}},\ldots,y_{j_{k+m}}\}$(where the x_l and y_l are listed in the order of their first appearence in φ and ψ, respectively), then $[\![\varphi\,\&\,\psi]\!]^M$ can be obtained from $\mathfrak{A}_{n,m}([\![\varphi]\!]^M,[\![\psi]\!]^M)$ by application of:[6]

$$\mathfrak{O}_{\varphi,\psi}(R) = \{(u_1,\ldots,u_n,v_{jk+1},\ldots,v_{jm}) \in \mathcal{U}^{n+m-k}\,|(u_1,\ldots,u_{n+m}) \in R\}.$$

Again, it may be noted that $\mathfrak{O}$ can be reduced to the more basic combinatorial operations $\mathfrak{J}$ and $\mathfrak{S}$, plus the (n-ary) *reflexivization*[7] defined by:

$$\mathfrak{R}_n(R) = \{(u_1,\ldots,u_n)|\ (u_1,\ldots,u_n,u_n)\in R\},$$

for any n+1-place relation R: $\mathfrak{O}_{\varphi,\psi}$ is of the form $\mathfrak{Z}_1\circ\ldots\circ\mathfrak{Z}_k$, where k is still the number of variables common to φ and ψ and each $\mathfrak{Z}_i$ is of the form $\mathfrak{J}^{k_i}\circ(\mathfrak{S}_{n_i}\circ\mathfrak{J}_{n_i})^{l_i}\circ\mathfrak{R}\circ\mathfrak{J}^{m_i}$; the verification of the messy details are kindly left to the reader.

3 I am indebted to Julia Hockenmaier and Klaus Robering for (independently) spotting an error in an earlier formulation of this operation.

4 The term 'product' is a misnomer if we are dealing with non-associative n-tuples; Bernays (1959: 3) thus proposes to view relations as sets of strings (*Kolonnen*).

5 Fr(φ) is the set of variables with free occurrences in φ.

6 I am indebted to Paul Dekker for spotting an error in an earlier formulation of this operation.

7 The term, due to Quine (1960), suggests that reflexive pronouns can be thought of as expressing $\mathfrak{R}_2$ operating on the extensions of transitive verbs.

We may thus conclude that first-order logic receives an almost compositional interpretation when understood as a logic of relations, whose formulae denote (or express) relations, and whose expressive means are there to combine these denotations. However, it seems that slight modifications in our conception of compositionality, or in the language of predicate logic, would be necessary to really make compositionality work: either the expressive means of predicate logic do not uniquely correspond to semantic operations, or else certain combinatorial operations on relations must be added to the ordinary logical operations of first-order logic.

1.2 Quine's Predicate Functors

The above compositionality considerations are a natural starting point for a semantic journey from ordinary first-order logic to its variable-free counterpart *PFL*. For the latter is a logic of relations containing, as its basic logical material, Boolean combination ($\mathfrak{N}$, $\mathfrak{A}$), quantification ($\mathfrak{E}$), as well as the basic combinators $\mathfrak{J}$, $\mathfrak{S}$, and $\mathfrak{R}$. Its syntax and semantics are easily given. It can be defined as a family $(PFL_n)_{n\in\omega}$ where each member contains the n-place predicates that are built up from the predicate constants of first-order logic by means of the *predicate functors* **N**, **A**, **E**, **J**, **S**, and **R** in the obvious way: if $P\in PFL_n$, then so are '**N***P*', '**J***P*', and '**S***P*'; if $P\in PFL_{n+1}$, then '**E***P*', and '**R***P*' are in PFL_n; and if $P\in PFL_n$ and $Q\in PFL_m$, then '**A***PQ*'$\in PFL_{n+m}$. A *model M (for PFL)* is just a first-order model assigning appropriate extensions to the basic predicate constants. And the *M-denotation* $\|P\|^M$ of any $P\in PFL_n$ is determined compositionally: $\|\mathbf{N}P\|^M = \mathfrak{N}_n(\|P\|^M)$, when $P\in PFL_n$, $\|\mathbf{A}PQ\|^M = \mathfrak{A}_{n,m}(\|P\|^M,\|Q\|^M)$, when $P\in PFL_n$ and $Q\in PFL_m$, etc. It is then clear from the above remarks, that every first-order formula φ (with n free variables) is equivalent to a predicate $\varphi^*\in PFL_n$ in the sense that $[\![\varphi]\!]^M = \|\varphi^*\|^M$, for any model M. To give an example, the composition formula (C) from section 1.1 could be expressed by the following binary *PFL* predicate:

(C*) **ESJJJ JRJJJA** *RS*,

where the first block of functors corresponds to the quantifier prefix '$(\exists y)$', and the second results from conjoining '$R(x,y)$' and '$S(y,z)$'.

The expressibility of *PFL*-predicates in first-order logic is equally straightforward. With each $P\in PFL_n$, we can associate a first-order formula P_* whose free variables are $\{x^1,\ldots,x^n\}$ (in that order), such that:

(!) $\quad M \models^{x_1,\ldots x_n}_{u1,\ldots u^n} P^* \text{ iff } (u_1,\ldots,u_n) \in \|P\|^M,$

for any model M. We let R_* be '$R(x^1,\ldots,x^n)$', when R is an n-ary predicate constant, and put:

$$
\begin{aligned}
(\mathbf{N}P)_* &= \text{'}\neg P_*\text{'};\\
(\mathbf{A}PQ)_* &= \text{'}[P_* \;\&\; Q^* \;[x^1,\ldots,x^m/x^{n-1},\ldots,x^{n+m}]]\text{'};\\
(\mathbf{E}P)_* &= \text{'}(\exists x_n)\; P_*\text{'};\\
(\mathbf{J}P)_* &= \text{'}[[P_*\vee\neg P_*] \;\&\; P_* \;[x^1,\ldots,x^n/x^n,x^1,\ldots,x^{n-1}]]\text{'};\\
(\mathbf{S}P)_* &= \text{'}[[P_*\vee\neg P_*] \;\&\; P_* \;[x^{n-1},x^n/x^n,x^{n-1}]]\text{'};\\
(\mathbf{R}P)_* &= \text{'}P_*[x^n/x^{n-1}]\text{'},
\end{aligned}
$$

whenever $P \in PFL_n$, and $Q \in PFL_m$.[8] (!) can then be proved by induction on P's complexity, but not here. It is not hard to see that applying the procedure $_*$ to (C*) results in a conjunction of eight formulae the first seven of which are tautologies, whereas the last conjunct is an alphabetic variant of (C); moreover, the variables occur in their 'natural' order, so that $[\![(C^*)_*]\!]^M$ is indeed $\|(C^*)\|^M$, whichever M we pick. On the other hand, by attaching the predicate functor **S** (or **J**) to the left of (C*), we obtain a translation of the form

(SC) $\quad [\tau(x^1,x^2)\ \&\ (\exists x^3)[\ldots[\ldots\ \&\ [R(x^2,x^3)\ \&\ S(x^3,x^1)]\ldots]],$

where $\tau(x^1,x^2)$ is a tautology with free variables x^1 and x^2 (in that order), and '...' indicates a conjunction of tautologies.

1.3 Second-order Logic

Let me now turn to *second-order predicate logic* by which, following Došen (1988), I shall mean the language consisting of individual and n-place predicate variables (for arbitrary n), Boolean connectives, and quantification over (individual and predicate) variables. The formation rules are the obvious ones; the *type* of an n-place predicate variable is n, that of an individual variable is 0. Note that we do not have any predicate or individual constants, nor do we have functors or identity. All these assumptions are made to simplify matters, but they do not affect the substance of the considerations to come.

Semantically, a formula like (2) can be interpreted with respect to an assignment **G** mapping individual variables on individuals, and n-place predicate variables on n-place relations over a given domain U:

(2) $\quad \neg(\exists R)(\forall x)[P(x) \rightarrow [R(x,y) \leftrightarrow \neg R(x,x)]].$

(2) is true under **G**, iff no binary relation R connects $\mathbf{G}(y)$ to exactly those elements of $\mathbf{G}(P)$ that do not bear R to themselves; hence (2) is a pompous formulation of the elementary predication '$P(y)$'. We thus see that (2) expresses a 'mixed' relation between individuals and sets of individuals: $\{(u,P) \in U \times \wp(U) \mid u \in P\}$. More generally, given a model M consisting of a universe U (and nothing else because there are no constants), a second-order formula φ expresses a relation $[\![\varphi]\!]^M$ between members of the sets U_n, where U_0 is U and U_n is $\wp(U^n)$ when $n \geq 1$:

(*) $\quad [\![\varphi]\!]^M =$

$$\{(u_1,\ldots,u_{n_0},P^1_1,\ldots,P^1_{n_1},\ldots,P^{m\varphi}_1,\ldots,P^{m\varphi}_{n_{m\varphi}}) \in U_0^{n_0} \times U_1^{n_1} \ldots \times U^{n_{m\varphi}}_{m\varphi} \mid$$

$$M \models \begin{matrix} x_1,\ldots,x_{n_0}\,R^1_1,\ldots,R^1_{n_1},\ldots,R^{m\varphi}_1,\ldots,R^{m\varphi}_{n_{m\varphi}}) \\ u_1,\ldots,x_{n_0}\,P^1_1,\ldots,P^1_{n_1},\ldots,P^{m\varphi}_1,\ldots,P^{m\varphi}_{n_{m\varphi}}) \end{matrix} \varphi\},$$

where $m\varphi$ is the largest -arity represented in φ, and for each $k \leq m\varphi$, φ contains n_k free variables of type k, listed in the order of their first free appearance in φ. The con-

[8] '$[\vec{x}/\vec{y}]$' indicates simultaneous substitution of free $\vec{x}$ by $\vec{y}$; the tautological conjuncts in the clauses concerning **J** and **S** are there to preserve the order of appearance.

tent of the above equation is quite straightforward: a second-order formula expresses the relation holding among the individuals and relations satisfying it. Its messiness arises from the fact that the exact order of arguments must somehow be determined: the order chosen here gives priority to the types of variables over their order of appearance; the obvious alternative will be discussed in section 3.1.

Given this notion of expressing mixed equations, it is natural to try and find a combinatory version of second-order logic along the lines of *PFL*. In fact, it seems that only little has changed in passing from first-order to second-order predicate logic, because the expressive means of the latter and their possible combinations are merely special cases of predicate logic in general. This point becomes more obvious when we think of second-order logic as a notational variant of *many-sorted* predicate logic, albeit with a heavily restricted interpretation (= class of models): generalizing the above considerations to the many-sorted case cannot be more than a technical exercise, and its result should then carry over to second-order logic as just defined.

I believe that this line of reasoning is essentially correct, even though it needs some elaboration. I would also like to argue that this is the most natural procedure for finding a combinatory version of second-order logic. This second point will be addressed in the final part. The details of the analogy between first-order predicate logic as a logic of relations and *PFL* on the one hand, and second-order logic and its combinatory counterpart on the other, will be the main objective of the next section.

2 A Variable-free Formulation of Second-order Logic

2.1 Categories

Although both predicate logic and predicate-functor logic are purely relational, the syntax of the latter will be given in terms of (classical) categorial grammar.[9] So we are given a set *Cat*, of *categories* derived from the *basic* categories t (≈ closed formula), and e (≈ individual term) by means of the binary operation /: if $a, b \in Cat$, then $a/b \in Cat$. The intended interpretation is the usual categorial one: a/b is the category of functors taking expressions of category b (to their right) as arguments, and yielding expressions of category a. The only rule of combination for building up complex expressions, is the principle of *Functional Application*: XY (= the result of concatenating X and Y) is of category a, whenever X is of category a/b and Y is of category b. Finally, for notational convenience, we allow ourselves a *neutral* category ε (of the *empty* string) such that $a/\varepsilon = \varepsilon/a = a$, for any category a.

As always in categorial grammar, understanding the members of *Cat* as syntactic classes is not the only interpretation possible. In particular, one may also think of *Cat* as a family of (functional) *types* of possible denotations. Thus t naturally corresponds to truth-values, e to individuals, a/b to functions from b type objects to a type objects,

[9] The qualification 'classical' is to distinguish Ajdukiewicz's (1935) syntax of functional application from its more sophisticated modern successors, like Lambek's (1958) calculus, as used by Došen (1988).

and ε to {∅} (or some other fixed singleton). Although we will not employ this interpretation officially, it is still important to keep it in mind, if only to connect the content of this part with what has been said before.

Among the infinity of categories, some are particularly interesting for our purposes and thus deserve their own terminology and notation. First of all, we may introduce a numerical notation for *first-order categories*: $\overline{0} = t$, $\overline{n+1} = \overline{n}/e$. Under the common identification of sets with their characteristic functions, $\overline{n}$ can be thought of as the category of n-place *first-order relations*, i.e. relations among individuals; note that $\overline{0}$ (= t) is the first-order category of closed formulae, expressing truth-values. In section 1.3 it was suggested that second-order formulae express 'mixed' or *second-order relations* of categories of the general form: $(\ldots(\overline{n_0}/\overline{1})/\ldots)/\overline{1})/\overline{2})\ldots/\overline{2})\ldots\overline{m})/\ldots)/\overline{m})$ where, for each i from 1 to m, $\overline{i}$ occurs n_i times. We may therefore represent second-order relations by finite sequences (= $(n_0, n_1, \ldots, n_m)$), where some of the n_i, but not n_m, may be 0. More precisely, the category $\overline{\sigma}$ is defined by recursion on the length $lg(\sigma)$ of non-empty finite sequences σ of natural numbers: $\overline{(n)} = \overline{n}$, $\overline{\sigma\hat{}0} = \overline{\sigma}$, and $\overline{\sigma\hat{}n+1} = \overline{\sigma\hat{}n}/\overline{lg(\sigma)}$; and we will usually identify σ and $\overline{\sigma}$. Categories of the form $(n_i)_{0\leq i\leq m}$ are called *second-order categories*. Note that every first-order category is also a second-order category.[10] As an example we may think of (1,1), i.e., $\overline{(1,1)}$, as the category of relations between individual terms and unary predicates, corresponding to functors combining with the latter to yield functors that need the former to make a formula: $\overline{(1,1)} = \overline{(1,0)}/\overline{1} = \overline{1}/\overline{1} = (t/e)/(t/e)$.

2.2 Predicate Functors

The main task in designing a combinatory version of second-order logic, is to provide a list of predicate functors and their categories determining their syntactic behaviour (given a suitable categorial apparatus). In designing this list, we will employ the analogy between first and second order. In particular, we will continue to think of predicate logic as a logic of relations, and of its basic means of expressions as operating on the relations expressed by its formulae. We may thus expect to have three kinds of predicate functors: (i) those that correspond to the basic means of expression already present in first-order predicate logic; (ii) those that correspond to second-order quantification; and (iii) the combinators making up for the loss of variables. Moreover, we may expect a close analogy between the latter and the combinators **J**, **S**, and **R** of *PFL*: an analogy but obviously no identity, for second-order logic has more variables to be moved around and the movement underlies additional syntactic restrictions: individual variables cannot be identified (via **R**, say) with 2-place relations, etc. So we shall expect slightly restricted versions of the old combinators but nothing more; and this is what we will have. As to the functors of group (i), we should not expect them to be identical to those we had had in first-order *PFL*: the latter were there to modify

[10] The terminology should be taken with care: a first-order relation is not necessarily one that is first-order *definable*; nor is a second-order relation one that is expressed by some second-order formula. Rather, the terms mark a distinction between the *types* of relations *generally expressed* by formulae of different order. In particular, there are first-order relations (like identity) that can only be expressed by second-order formulae.

first-order relations, whereas now we need something to operate on *second-order* relations. However, we would expect these operators to be quite analogous to **N**, **A**, and **E**, as indeed they will be. Finally, second-order quantification should be a variant of **E**, operating on second-order relations, and reducing different kinds of argument-places than the second-order counterpart of **E**. All of this may sound trivial and obvious, but it is precisely these analogies between first and second order that are absent from Došen's (1988) version of second-order predicate functor logic.

Let us start with the predicate functors under (i). As in *PFL*, negation will be represented by an operator turning one relation into another one of the same type. We thus assume that the symbol '**N**' appears in all (and only the) categories of the form σ/σ, where σ is a second-order category. Note that this covers the ordinary first-order uses of **N** in *PFL*: they turn out to be special cases. The same will be true of all predicate functors of second-order logic. Needless to say, the intended interpretation of negation is as before, i.e. complementation.

Conjunction is similar: if σ and τ are second-order categories, **A** will appear (only) in every category of the form $((\sigma+\tau)/\tau)/\sigma$, where addition on finite sequences is pointwise: $(n_i)_{0\leq i\leq k} + (m_i)_{0\leq i\leq l} = (n_i + m_i)_{0\leq i\leq max(k,l)}$. Again the intended interpretation should be obvious: the result $R \oplus R'$ of conjoining two second-order relations R and R', consists of all tuples of the form $\overrightarrow{X_0}\,\overrightarrow{Y_0}\ldots\overrightarrow{X_{max(k,l)}}\,\overrightarrow{Y_{max(k,l)}}$, where $\overrightarrow{X_0}\ldots\overrightarrow{X_l}\in R$, $\overrightarrow{Y_0}\ldots\overrightarrow{Y_k}\in R'$, juxtaposition indicates tuple-concatenation, $\overrightarrow{X_j}\in U_j^{m_j}$, and $\overrightarrow{Y_j}\in U_j^{m_j}$, for all $j\leq max(k,l)$.

Quantification over individuals is equally straightforward to adapt: the predicate functor **E** applies to any relation among $n+1$ individuals and m relations of various -arities, turning it into a relation between n individuals and the same relations. We thus assume that the predicate functor **E** occurs in all, and indeed only, the categories of the form (σ/σ'), where σ and σ' are second-order categories that only differ in $\sigma_0 = \sigma'_0 - 1$.

Now for (ii) second-order quantification. Since a formula of second-order predicate logic may contain more than one predicate variable, there is in general more than one way of applying a second-order quantifier to it. In the case of first-order quantification, all of these combinations correspond to a single *PFL*-operator, **E**, plus a suitable permutation of variables expressed by a block of **J**s and **S**s: **E**'s quantificational force only affects the final argument position. Such a simple correspondence is not available for second-order quantification: the last argument position of a second-order relation is, by definition, one of a maximal -arity. In order to also be able to affect the argument positions occupied by smaller relations, we obviously need more than just one second-order counterpart of **E**.[11] Indeed, we need an $\mathbf{E}_i$ for each -arity i, operating on the last argument position occupied by i-place relations; the rest will again be done by combinatorial predicate functors. So $\mathbf{E}_i$ turns a second-order relation of category $(n_0,\ldots,n_i+1,\ldots,n_m)$ into its projection of category $(n_0,\ldots,n_i,\ldots,n_m)$. If we now identify the first-order functor **E** as $\mathbf{E}_0$, we get a general scheme of quantification in second-order predicate functor logic: if $i\geq 0$ is a natural number, the predicate functor $\mathbf{E}_i$ occurs in all and only the categories of the form (σ/σ'), where σ and σ' are second-order cate-

[11] Actually, the multiplication of quantifiers is due to our particular definition (*) of satisfaction: see section 3.1 for an alternative.

gories that only differ in that $\sigma_i = \sigma'_i - 1$. The intended interpretation is as expected: the result of modifying R by the i-th existential quantifier is the set of all tuples $(\overrightarrow{X_0}, \ldots, \overrightarrow{Y_m})$ such that there is some $Y \in U_i$ for which $(\overrightarrow{Y_0} \ldots \overrightarrow{Y_m}) \in \mathcal{R}$, where $\overrightarrow{Y_i} = \overrightarrow{X_i}, Y$, and $\overrightarrow{Y_j} = \overrightarrow{X_j}$ whenever $j \neq i$.

So only the predicate functors of the third kind, i.e. suitable second-order versions of *PFL*'s combinators **J**, **S**, and **R**, remain to be specified. The above discussion has made it clear that they, too, must be indexed with the -arities upon which they act. Thus, if $i \geq 0$ is a natural number, we have second-order predicate functors $\mathbf{J}_i$, $\mathbf{S}_i$, and $\mathbf{R}_i$. The first two appear in the functor categories (σ/σ), where σ is a second-order category of length $\geq i$; and $\mathbf{R}_i$ will be found in precisely the categories of $\mathbf{E}_i$. The specification of the intended meanings of these functors is left to the reader's imagination.

Table 1 gives a complete specification of the family $F = (F_{a \in Cat})$ of second-order predicate functors. The notation '$\sigma \approx_i \sigma''$' means that σ and σ' are of the same length, and $\sigma_j = \sigma'_j$ whenever $j \neq i$; σ and τ are arbitrary second-order categories, and 'i' ranges over natural numbers.

Functors	*Categories*
N	σ/σ
A	$((\sigma + \tau)/\sigma)/\tau$
$\mathbf{E}_i$	σ/σ': $\sigma \approx_i \sigma'$, $\sigma_i = \sigma'_i - 1$
$\mathbf{J}_i$	σ/σ': $\sigma_i \geq 2$
$\mathbf{S}_i$	σ/σ': $\sigma_i \geq 2$
$\mathbf{R}_i$	σ/σ': $\sigma \approx_i \sigma'$, $\sigma_i = \sigma'_i - 1$

*Table 1: The family **F** of second-order predicate functors*

2.3 Elementary Predication

Table 1 cannot be the complete story concerning second-order predicate functor logic: the second-order categories are empty and categorial combination obviously does not fill them. To get the categorial machinery off the ground, the predicate functors need something to operate on. In the case of first-order logic, the (first-order) predicate constants served as arguments to the (first-order) predicate functors. In second-order logic, we do not have any predicate constants, so what shall the functors in Table 1 be applied to? Whatever it is, it will have to correspond to the atomic formulae of second-order logic, just like first-order predicates represent atomic first-order formulae in PFL. Now, the atomic formulae of second-order logic are all of the form

(3) $\qquad R(x_1, \ldots, x_n),$

where the x_i are individual variables and R is an n-ary predicate variable. We may assume that the variables occurring in (3) are pairwise distinct, any repetitions being dealt with by functors **J**, **S**, and **R**. According to our scheme of interpretation, (3) expresses the (n+1-ary) second-order relation of *(elementary) predication* that holds

among individuals $u_1, \ldots, u_n$ and n-ary first-order relations R, if and only if, R holds among $u_1, \ldots, u_n$. To stress the point, we may even reformulate (3) in a two-sorted fashion, using **PRED** as a second-order predicate constant:

(3′) $\quad$ **PRED** $(R, x_1, \ldots, x_n)$

Of course, (3′) is not a formula of (our version of) second-order logic, but it does reveal the logical form of (3) in terms of a given relation of elementary predication.[12] Under our distinctness assumption, the formula (3′) – viewed as a first-order formula – would be represented by **PRED** alone: it expresses what **PRED** denotes. It is therefore natural to include among the basic means of expression of second-order predicate functor logic, a second-order predicate expressing elementary predication.

So, apart from the more or less obvious generalizations of the *PFL*-functors, second-order predicate functor logic contains an additional logical tool, viz. predication. Indeed, we need a whole family of symbols $\mathbf{PRED}_n$ ($n \geq 1$), but their interpretation is completely schematic. Note that these symbols are not second-order predicate constants in the ordinary sense: they are logical, not contingent (model-dependent). Nor are they predicate functors: they denote relations, not modifiers. What is their category? Obviously that of the relation expressed by (3), i.e. the category abbreviated by the n+1-place sequence $(n,0,\ldots,0,1)$, which turns out to be $\overline{n}/\overline{n}$, which itself is a category of *first-order* predicate functors. Indeed, the functional analogue of n-ary elementary predication is the *identity mapping* over the domain of n-place first-order relations.

2.4 Translation

I will now sketch a translation procedure assigning to every second-order formula, a semantically equivalent formula of second-order predicate functor logic. The section will not contain anything new, but simply give a more precise account of what has been done so far. Moreover, my aim is a very modest one, viz. to prove the mere existence of a semantically correct interpretation of second-order logic in terms of the predicate functors in *F*. In particular, I am not interested in questions of elegance or complexity. Finally, I will not say anything about the reverse direction from *F* to second-order logic; this is partly because I take it to be even more obvious and closer to the first-order case, and partly because I will return to these matters in the final part. These things said, let me start with some basic definitions surrounding the semantic effects of second-order predicate functors.

A finite sequence of natural numbers is called a *trivial derivation* if it is an identical mapping $\overset{n}{\mathbf{i}}$, i.e. of the form $(0,\ldots,n-1)$. Let $n \geq 2$ be a natural number. Then the *n-jump* is that finite sequence $\overset{n}{\mathbf{j}}$ of length n that satisfies: $\overset{n}{\mathbf{j}}(0) = n-1$ and $\overset{n}{\mathbf{j}}(m+1) = m$ whenever $0 \leq m \leq n-2$; the *n-swap* is that n-place sequence $\overset{n}{\mathbf{s}}$ such that $\overset{n}{\mathbf{s}}(n-2) = n-1$, $\overset{n}{\mathbf{s}}(n-1) = n-2$, and $\overset{n}{\mathbf{s}}(m) = m$, whenever $0 \leq m < n-2$; and the *n-reflex* is the n-place sequence defined by: $\overset{n}{\mathbf{r}}(n-1) = n-2$ and $\overset{n}{\mathbf{r}}(m) = m$ if $0 \leq m < n-1$. A *basic derivation* is a

[12] (3′) can be either read as a two-sorted first-order formula, or as a formula of second-order logic containing the (second-order n+1-ary predicate) constant **PRED**. In both cases, (3′) is only equivalent to the second-order formula (3) if we restrict attention to 'standard' models in which **PRED** denotes elementary predication.

sequence of the form $\overset{y}{\mathbf{i}}, \overset{y}{\mathbf{j}}, \overset{n}{\mathbf{s}}$, or $\overset{n}{\mathbf{r}}$. A *derivation* is a finite sequence of (not uniquely) the form $\delta = \beta_1 \circ \ldots \circ \beta_k$, where all β_i are basic derivations. An arbitrary finite sequence over a non-empty set $\mathcal{U}$ is *free*, iff it is injective, i.e. if it contains no repetitions. The combinatorial fact underlying the adequacy of first-order *PFL* can be stated as follows:

(!) If σ is a finite sequence and τ is a free sequence such that $\mathrm{rge}(\tau) = \mathrm{rge}(\sigma)$, then $\sigma = \tau \circ \delta$, for some derivation δ.

(!) is easily proved by induction on τ's length, given certain obvious properties of basic derivations. The verification of the details is left to the reader. The way in which (!) leads to a combinatorial formulation of first-order logic is equally straightforward, and can be seen as a special case of the following adaptation to the second-order situation.

For any non-empty set U and positive integer n, we let U_n be the power set of the Cartesian product U^n, and put: $U_* = \bigcup_{n \in \omega} U_n$, where $U_0 = U$. A *satisfier* Σ $(= (\Sigma_i)_{0 \leq i \leq n})$ *over* such U is a finite sequence of finite sequences over U_*, such that each Σ_i consists of members of U_i and $\Sigma_n \neq \emptyset$; we will refer to the sequence $(lg(\Sigma_0), \ldots, lg(\Sigma_n))$ as Σ's *type*. Σ is *free* iff each Σ_i is a free sequence. A *Basic Derivation* Δ is a finite sequence of basic derivations, all but (exactly) one of which are trivial. If Δ and Δ' are finite sequences of derivations of the same type, then $\Delta \square \Delta'$ is the sequence $(\Delta_i \circ \Delta_i')_{0 \leq i < lg(\Delta)}$. A *Derivation* is a finite sequence $\Delta = \mathrm{B}_1 \square \ldots \square \mathrm{B}_k$, where all B_i are basic Derivations of the same type. The natural second-order counterpart of (!) is:

(!!) If Σ is a satisfier and T is a free satisfier such that $\mathrm{rge}(\Sigma_i) = \mathrm{rge}(\mathrm{T}_i)$ for all $i \leq lg(\Sigma)-1$, then $\Sigma = \mathrm{T} \square \Delta$, for some Derivation Δ.

(!!) immediately follows from (!): Δ can be constructed by gradually transforming each T_i into Σ_i, using a basic derivation β_i obtained from (!), and expanding it to a Basic Derivation B_i in the only possible way. We may now use (!!) to reduce the relations expressed by arbitrary quantifier-free second-order formulae to the relations expressed by the particularly simple and easily translatable *free* formulae, in which each variable occurs exactly once. The following two observations make the connection:

(a) If x and y are variables of the same type, φ is a second-order formula, y occurs freely in φ but not in the scope of $(\exists x)$, then there is a Derivation Δ such that $[\![\varphi[y/x]]\!]^M = \{\Sigma \mid \Sigma \square \Delta \in [\![\varphi]\!]^M\}$, for every model M.

(b) If P is a second-order predicate and Δ is a Derivation, then there is a second-order predicate Q such that $\|Q\|^M = \{\Sigma \mid \Sigma \square \Delta \in \|P\|^M\}$ for every M.

(a) is a direct consequence of what is sometimes called the *Substitution Lemma*: Σ satisfies $\varphi[{}^{y}/_{x}]$ iff Σ' satisfies φ, where Σ' is a certain satisfier with Σ's range. – (b) is a direct application of the interpretation of (second-order) predicate functors; it can be proved by induction on the structure of Derivations.[13]

We can use (a) and (b) to define a translation *: $\varphi \mapsto \varphi^*$ satisfying:

(c) If φ is a second-order formula, then there is a second-order predicate functor expression φ^* such that $[\![\varphi]\!]^M = \|\varphi^*\|^M$, for any model M.

[13] More accurately, the induction would have to run on the length of *names* of Derivations, defined in the way suggested by the above notation.

* can be defined by induction on φ's syntactic complexity. An atomic φ may be obtained from a free φ', by replacing free (individual) variables so that, by iterated application of (a) and the associativity of ▫, we may conclude that $[\![\varphi]\!]^M$ is of the form $\{\Sigma|\,\Sigma \circ \Delta \in [\![\varphi']\!]^M\} = \{\Sigma|\,\Sigma \circ \Delta \in \|\mathbf{PRED}_n\|^M\}$; so (b) applies, providing us with a predicate functor version of φ. A similar argument works in the case of conjoined formulae [φ & ψ], which can be obtained from free [φ' & ψ'] of equal complexity by substituting free (individual and predicate) variables; by a renaming of bound variables (to make (a) applicable) and the inductive hypothesis, we have:

$$\begin{aligned} \mathrm{MMM}[\![\varphi \,\&\, \psi]\!]^M &= \{\Sigma|\,\Sigma \circ \Delta \in [\![\varphi' \,\&\, \psi']\!]^M\} = \{\Sigma|\,\Sigma \circ \Delta \in [\![\varphi']\!]^M \oplus [\![\psi']\!]^M\} \\ &= \{\Sigma|\,\Sigma \circ \Delta \in \|\varphi'^*\|^M \oplus \|\psi'^*\|^M\} = \{\Sigma|\,\Sigma \circ \Delta \in \|\mathbf{A}\ \varphi'^*\ \psi'^*\|^M\}, \end{aligned}$$

so that (b) applies again. Negation being straightforward, we are left with quantified formulae $(\exists x)\varphi$, where x is a variable of type n. By the combinatorial fact (!!),

$$\Sigma \in [\![(\exists x)\varphi]\!]^M \text{ iff } \Sigma^U \circ \Delta \in [\![\varphi]\!]^M,$$

for some $X \in U_n$, where Σ^X is like Σ except that Σ_n^X extends Σ_n by X, and where Δ is a fixed Derivation determined by x's first appearance in φ. Thus, using induction, we know that $[\![(\exists x)\varphi]\!]^M = \{\Sigma|\,\Sigma^X \circ \Delta \in \|\varphi^*\|^M$, for some $X \in U_n\}$

$$= \{\Sigma|\,\Sigma^X \circ \Delta \in \|\varphi^*\|^M, \text{ for some } X \in U_n\} = \{\Sigma|\text{ for some } X \in U_n\colon \Sigma^X \in [\![Q]\!]^M\},$$

for some second-order predicate functor expression Q provided by (b). Hence we let $((\exists x)\varphi)^*$ be $\mathbf{E}_n Q$.

3 Alternative Formulations

3.1 Subscripts Explained Away

In spite of the schematic character of Table 1, the infinity of logical operators and combinators may be regarded as a serious defect of our combinatory formulation of second-order logic.[14] It is therefore natural to ask whether we cannot do better, and somehow find a finite reformulation of the above system. Now, although I have not been able to arrive at this goal (nor to prove the impossibility of this enterprise), it turns out that a simple modification in our basic *semantic* concepts almost gets us there: it allows us to reduce Table 1 to a finite list, leaving us with the infinity of the **PRED** symbols. In this section, I will briefly sketch this almost finite version of second-order predicate functor logic.

In section 2.3 we have used two-sorted first-order logic in order to explain away the variables of atomic formulae. We could have gone a step further and derive combinatorial translations φ^0 of arbitrary second-order formulae φ, by simply transforming their two-sorted analogues $\tilde{\varphi}$, using standard devices of first-order *PFL*, as dis-

[14] It shares this defect with Došen's (1988) version, even though (in section 3) the latter discusses some possible reductions of his logical operators by categorial means, but none of them leads to a finite list.

cussed in section 1. Of course, this procedure would not have been in accord with our definition (*) of second-order satisfaction, which gives priority to the variables' types over their order of appearance. But, certainly, we could have given an equally adequate and plausible 'type-free' definition of satisfaction:

$$(^{*}_{*}) \qquad \|\varphi\|^{M} = \{(X_1, \ldots, X_n) \in U_*{}^n \mid M \models {}^{x_1, \ldots, x_n}_{X_1, \ldots, X_n} \varphi\},$$

where f contains the (individual and predicate) variables $x_1, \ldots, x_n$, in that order. In fact, $(^{*}_{*})$ is essentially a special case of the concept of satisfaction in first-order logic. And, clearly, using $\|\cdot\|$ instead of $[\![\cdot]\!]$ would render the translation $\varphi \overset{1}{\rightsquigarrow} \tilde{\varphi} \overset{2}{\rightsquigarrow} \varphi_0$ fully adequate. Moreover, it is obvious that this translation can do without any subscripts on predicate functors, because they are not needed in step **2**.

Instead of following this line of reasoning down to its details, let us explore its consequences by looking at some examples. Consider the (contradictory) second-order formula (4) and its two-sorted analogue $(\tilde{4})$:

(4) $\quad \neg(\exists P)\, \neg(\exists x)\, P(x)$

$(\tilde{4})$ $\quad \neg(\exists P)\, \neg(\exists x)\, \mathbf{PRED}_1(P,x)$

$(\tilde{4})$ and, consequently, (4) can be given the following *PFL*-translation:

(4^0) $\quad \mathbf{N\,E\,N\,E\,PRED}_1$.

If we compare this result with that of the procedure * from section 2.4, we find a striking similarity:

(4*) $\quad \mathbf{N\,E}_2\,\mathbf{N\,E}_1\,\mathbf{PRED}_1$

But the similarity is deceiving, because it depends on the fact that the innermost quantifier in (4) happens to bind the last variable in its matrix. This becomes apparent when we start moving the quantifiers in (4), turning the contradiction into a tautology:

(5) $\quad \neg(\exists x)\, \neg(\exists P)\, P(x)$

$(\tilde{5})$ $\quad \neg(\exists x)\, \neg(\exists P)\, \mathbf{PRED}_1(P,x)$

(5^0) $\quad \mathbf{N\,E\,N\,E\,J\,PRED}_1$

(5*) $\quad \mathbf{N\,E}_1\,\mathbf{N\,E}_2\,\mathbf{PRED}_1$

The presence of **J** in (5^0) is forced by the fact that the inner quantification does not affect the last argument of the relation expressed by $\mathbf{PRED}_1$. No such device is necessary (and, in fact, applicable) in (5*), where the two argument positions of $\mathbf{PRED}_1$ are independently accessible, due to their distinct types.

The freedom with which the functors **J** and **S** apply to mixed relations like $\mathbf{PRED}_1$ in ^{0}-translations, but not in our *-version of second-order predicate functor logic, may give the wrong impression that the subscript-free variant of second-order logic is simpler, in that its syntax can be given without a rather intricate system of categories such as *F*. However, the correct use of the reflexive functor R depends on the type of the relation it is applied to. For, certainly, in neither variant of second-order *PFL* anything like $\mathbf{R\ PRED}_1$ is interpretable, despite the fact that predication is a binary relation. So we

may safely conclude that a subscript-free formulation of second-order predicate functor logic is as complicated as our version. But, of course, these complications are only due to the hidden *sortedness* of second-order logic, not to its semantic complexity in terms of higher-order quantification: it would also arise in the translation of many-sorted first-order logic along the lines of the present section.

3.2 Remarks on Došen's 'Second-order Logic without Variables'

We finally get to a brief comparison of the language of section 2, with Kosta Došen's (1988) version of second-order predicate functor logic. Ignoring details of implementation as well as Došen's strictly syntactic perspective, the most striking difference between the two approaches lies in Došen's use of a quite different kind of predicate functor that cannot be found in either Quine's original formulation, nor in the above versions of *PFL*. Let me give an idea of what I take to be Došen's reason for adopting these functors.

Došen's overall strategy for finding a variable-free formulation of second-order logic is to extend Quine's first-order *PFL* by adding new logical material, but without changing what is already there. Thus, e.g., negation should be translated by the *first-order* predicate functor **N** of category $\overline{n}/\overline{n}$, because it already appears in Quine's *PFL*. But this categorization obviously poses a problem when we want to negate *second-order* formulae. Take an atomic formula $P(x)$ expressing elementary predication, and thus to be translated by a (primitive) mixed predicate $\mathbf{PRED}_1$ of category $\overline{n}/\overline{n}$, as we have seen in section 2.3![15] But, certainly, we cannot combine the latter with Quine's first-order **N** by functional application; rather, what we need is *functional composition*:

$$
\begin{array}{ll}
 & [\![\neg P(x)]\!]^M\ (R)(u) = 1 \\
\text{iff} & [\![P(x)]\!]^M\ (R)(u) = 0 \\
\text{iff} & \|\mathbf{PRED}_1\|^M\ (R)(u) = 0 \\
\text{iff} & \mathfrak{N}_1(\|\mathbf{PRED}_1\|^M\ (R))(u) = 1 \\
\text{iff} & \mathfrak{N}_1 \circ \|\mathbf{PRED}_1\|^M\ (R)(u) = 1 \\
\text{iff} & \|\mathbf{C}(\mathbf{N},\mathbf{PRED}_1)\|^M\ (R)(u) = 1
\end{array}
$$

where **C** is a new operator expressing functional composition. Similar considerations show that we need a ternary variant of functional composition in order to conjoin second-order formulae by means of Quine's first-order **A**. Otherwise, Došen's second-order predicate logic is quite similar to the version sketched in section 3.1.

The inclusion of composition operators raises the question of whether they are actually needed. Došen's (1988: 256) answer is: 'We suppose that their presence in our variable-free formulation of this second-order language, shows in what way the combinatory role of bound variables in second-order logic is more complex than the combinatory role of bound variables in first-order logic'. I believe this conclusion to

[15] This analysis of atomic second-order formulae is in accord with Došen's account, where we find so-called 'combinatory predicate functors' Id_k. Unfortunately, Došen (1988: 255) does not make the connection between them and atomic second-order formulae expressing elementary predication and concludes that they 'are introduced for more technical reasons'.

be somewhat exaggerated. For, as we have seen, there are alternative formulations of combinatory second-order logic that can do without any operators that are not direct adaptations of Quine's original predicate functors.

One may object that, although it is technically possible to explain the composition operators away, with them we get a more natural or more plausible combinatory formulation of second-order logic. Second-order predicate logic is, after all, an extension of first-order logic, obtained by adding new quantifiers. So shouldn't second-order predicate functor logic be an *extension* of Quine's *PFL*, instead of a recategorization procedure? I think not. First of all, we may note that, in a sense, Došen's variable-free second-order logic also needs some recategorization of old predicate functors: it contains second-order versions of **J**, **S**, and **R**, and whether we call these new, second-order functors or recategorizations of old, first-order ones is merely a matter of taste and terminology. Secondly, it is not quite true to say that only quantifiers distinguish second-order logic from first-order logic: we also have new *free* variables denoting relations, and these variables play an essential role in the most natural semantic interpretation of second-order as a logic of (mixed) relations. Most importantly, however, the idea of using Quine's first-order functors in second-order logic is at odds with the original motivation behind *PFL*. Let me explain.

If first-order formulae express relations among individuals, then certainly second-order *PFL* should take second-order formulae as expressing relations among first-order relations and individuals. Now, because first-order formulae express relations, Boolean connectives operating on the meanings of such formulae are naturally interpreted as operators on such relations, i.e. as Quinean predicate functors; the same is true of the quantifiers. By the same token, a second-order version of *PFL* would naturally interpret Boolean connectives as operating on the meanings of second-order formulae, i.e. relations among individuals and relations; similarly, second-order *PFL* would have to regard quantifiers (of whatever order) as such higher-order predicate functors. In particular then, there is no place for Quine's original predicate functors in second-order logic! Rather, we would have to recategorize negation, conjunction, and existential quantification (as well as the purely combinatory operators) as higher-order functors—just as we did in Table 1 above.

The contrast between Došen's introduction of composition operators and our strategy of recategorizing old functors, is familiar from Categorial Grammar: instead of directly combining two expressions of categories a/b and b/c into something of category a/c by a principle of Functional Composition, we may *raise* the first one to category $(a/c)/(b/c)$ *by Geach's Rule*: whatever is of category a/b, is also of category $(a/c)/(b/c)$.[16] It is clear that, in the presence of the principle of Functional Application, Geach's Rule can be used to derive Functional Composition; moreover, in the absence of any higher-order functors, the latter is all that Geach's Rule can be used for. Let us look at our simple example $\neg P(x)$ again! The atomic formula is of category (1,1), i.e. $(t/e)/(t/e)$ and so is Quine's negation. On the other hand, in Table 1 we also

[16] See van Benthem (1991: 26) or von Stechow (1991: 124f.) for discussions on the relation between Geach's Rule and functional composition. The correspondence needed for our present purposes is actually slightly more general—as in Geach's (1970) original paper.

find negation **N** in category $(1,1)/(1,1) = [(t/e)/(t/e)]/[(t/e)/(t/e)]$, i.e. the result of applying Geach's Rule to **N**'s original category $a/b = (t/e)/(t/e)$ and $c = (t/e)$. From Došen's point of view, our categorization of **N** can thus be seen as a way to avoid a principle of Functional Composition or, equivalently, predicate functors to this effect.

Another application of Geach's Rule may be seen in Quine's original replacement of the truth-functional connective $\neg$ by the predicate functor **N**. From a categorial point of view, the former is of category t/t, whereas the latter's unary variant is in $(t/e)/(t/e)$, as we have just seen.[17] In section 1.1 it was argued that the semantic idea behind this type shift may be seen as a step towards compositionality: more complex meanings (relations instead of truth-values) call for more complex semantic combinations. Of course, this complexity could equally well have been achieved by adding composition operators, mediating between negation as a truth-functor and predicates it is applied to. That is, instead of $\mathbf{N}\varphi^*$, one could have given the combinatory version of $\neg\varphi$ as $\mathbf{C}(\neg, \varphi^*)$, where **C** is a predicate functor expressing functional composition: $\|\mathbf{C}(f,P)\|^M(u) = 1$ iff $\|f\|^M(\|P\|^M)(u) = 1$. Surely, this is not what Quine did. But the fact that he could have done it, shows that the composition operators in Došen's version of second-order *PFL* cannot be explained by a higher complexity of variable binding in second-order logic: we could have them in variable-free formulations of first-order logic, and they would do the very same job of avoiding a recategorization of operators. Any syntactic complications in second-order logic are due to its sorted character that is reflected, not in the need for new functors, but in the complicated assignment of categories to the old ones.[18]

Notes

… on the pre-published version Zimmermann (1996)

Since I finished this paper in 1993, my views on variable-free logics have undergone some serious changes, and are therefore not always adequately reflected in the above text; I have still decided to publish the material in its present form, because a thorough revision would presumably cost me more time than I can afford at the moment. With very few exceptions (one of which is noted in footnote 3), the corrections I have nevertheless made are due to observations made by Klaus Robering, without whose encouragement the paper would never have left my drawer; I hope to be able to reply to his more profound criticisms at a later opportunity.

… on the present version

The revision announced in the above note still has to wait for another opportunity. The present version is almost identical to its predecessor: with the exception of the one mentioned in footnote 6, only a few minor errors have been corrected. All changes result from suggestions made by Paul Dekker and Michelle Weir, both of whom I would like to thank for their careful reading.

[17] The possibility of generating predicate negation from a truth-functional connective was part of the motivation for Geach's (1970) proposal.

[18] This paper has profited from correspondence and discussions with Kosta Došen, who – despite our disagreement – encouraged me to turn what was originally planned as part of a review into an article. For critical remarks and various hints I am also grateful to Franz Beil, Regine Eckardt, Urs Egli, Fritz Hamm, audiences at the universities of Stuttgart and Tübingen and, most of all, Hans Kamp.

References

[Ajdukiewicz 1935] Ajdukiewicz, Kazimierz: *Die syntaktische Konnexität.* – In: Studia Philosophica 1 (1935), 1–27.

[Bacon 1985] Bacon, J.: *The completeness of a predicate-functor logic.* – In: Journal of Symbolic Logic 50 (1985), 903–926.

[van Benthem 1991] Benthem, Johan van: *Language in action.* – Amsterdam: North-Holland 1991.

[Bernays 1959] Bernays, Paul: *Über eine natürliche Erweiterung des Relationenkalküls.* – In: A. Heyting (ed.): Constructivity in Mathematics. – Amsterdam: North-Holland 1959, pp. 1–14.

[Došen 1988] Došen, Kosta: *Second-order logic without variables.* – In: W. Buszkowski et al. (eds.): Categorial Grammar. Amsterdam/Philadelphia 1988, pp. 245–264.

[Geach 1970] Geach, P. T.: *A program for syntax.* – In: Synthese 22 (1970), 3–17.

[Lambek 1958] Lambek, Joachim: *The mathematics of sentence structure.* – In: American Mathematical Monthly 65 (1958), 154–170.

[Quine 1960] Quine, Willard Van Orman: *Variables explained away.* – In: Proceedings of the American Philosophical Society 104 (1960), 343–347.

[Quine 1974]–: *Truth and disquotation.* – In: L. Henkin et al. (ed.): Proceedings of the Tarski Symposium. Providence 1974, pp. 373–384.

[Stechow 1991] Stechow, Arnim von: *Syntax und Semantik.* – In: D. Wunderlich, A. v. Stechow (eds.): Semantik/ Semantics. – Berlin: de Gruyter 1991, pp. 90–148.

[Zimmermann 1996] Zimmermann, Thomas Ede: *Second-order variables explained away.* – In: K. Robering (ed.): Kategorien und Funktoren in Syntax und Semantik. Institut für Linguistik, Arbeitspapier Nr. 32, Technische Universität Berlin 1996, pp. 121–141.